Sketch-based pruning of a solution space within a formal geometric constraint solver [1]

C. Essert-Villard [a] , P. Schreck [a] , J.-F. Dufourd [a]

[a] *Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection (LSIIT, UPRES-A CNRS 7005) Université Louis-Pasteur de Strasbourg Boulevard Sebastien-Brant 67400 Illkirch, France*

Abstract

In CAD systems, formal geometric solvers enable the designer to draw a sketch and to provide constraints that are compiled into a construction plan by symbolic geometric reasoning. Then the plan is interpreted in order to generate the required figure. In case there are multiple solutions, they allow to scan the entire solution space. But when the number of solutions becomes too high, it is very time-consuming to examine each of them to determine which one is the closest to the user's will. In this paper, we introduce a sketch-based heuristic that enables to easily eliminate most of the solutions and to keep, among a solution space represented by a tree, only one branch, or at the worst a small subtree of solutions, that has the best likeness with the original sketch.

Key words: Formal geometric constructions; Symbolic constraint solving; Tree pruning; Computer-aided design

1 Introduction

In Computer-Aided Design (CAD), drawing constrained figures and automatically solving them are subjects that have been studied by many authors [1,3–5,15,17,20,22,25], but that still remain topical. In geometric modeling, constraints bring a precise description of geometric properties that must be

[1] This research is supported by the PRC-GDR "Algorithms, Languages and Programming", MENRT-CNRS, France

respected by the object. Since I.E. Sutherland's Sketchpad [24], various ways of solving constraints have been considered. Two main classes can be distinguished: numerical approaches and formal methods. The first ones are often chosen, and consist in solving numerically the equation system related to the dimensions [12,16,17]. A formal resolution of the symbolic constraint system offers the advantage of efficiently manipulating the defined figure by varying parameters values [1,4,5,21]. Unfortunately, algebraic tools for formal calculus, e.g. computer algebra systems [11], are much too general to be efficient for that purpose.

Within our team, we gave priority to a both formal and geometric approach: resolution tools that we propose are based on classic geometry, more particularly geometric constructions domain, and on symbolic reasoning, more precisely rule-based systems. They are specialized enough to be efficient to solve most problems in CAD. Then, in a first stage, the result of the formal resolution of a constraint system is a *construction plan* describing, in the right order, the objects to build and the operations to apply so as to obtain a solution figure. A second stage consists in numerically interpreting this construction plan, by replacing the parameters with dimensions values. This approach was materialized by a software called YAMS [2] [8,9,18] that associates a formal geometric solver with the 3D topology-based modeller *Topofil* [2].

Whatever the approach, numerical or formal, algebraic or geometric, a constraint system doesn't usually define a single figure. When an infinite number of figures satisfies the constraints, the system is said under-constrained. When the set of solutions is finite and non empty, the system is said *well-constrained*. In the case of a well-constrained system, the exploration of the solutions space is not as easy as it seems. Indeed, the existence of polynomial equations whose degree is higher or equal to 2, on an algebraic point of view, or of multiple intersections, on a geometric point of view, quickly leads to a combinatorial explosion of the number of solutions. As regards our approach, this increase in the number of solutions is the result of the existence of *multifunctions* in the construction plan.

As an example, Fig.1 shows a sketch made up of fifteen adjacent triangles. The lengths of all their sides are asked to be equal to a given dimension. This kind of configuration was studied by Owen [20], and is known to have 2^{p-2} distinct solutions, where p is the number of points. In our case, with 17 points, we obtain 32768 solutions (triangles are often superposed, because their sides are equal). Some of them are presented on Fig.2.

In most cases, CAD users only want one solution figure when they design an object. That's why an important matter of geometric solvers is identifying the

[2] YAMS: Yet Another Meta Solver

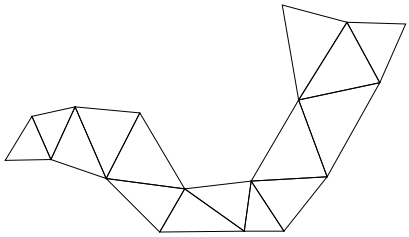


Fig. 1. 15 triangles configuration: the sketch

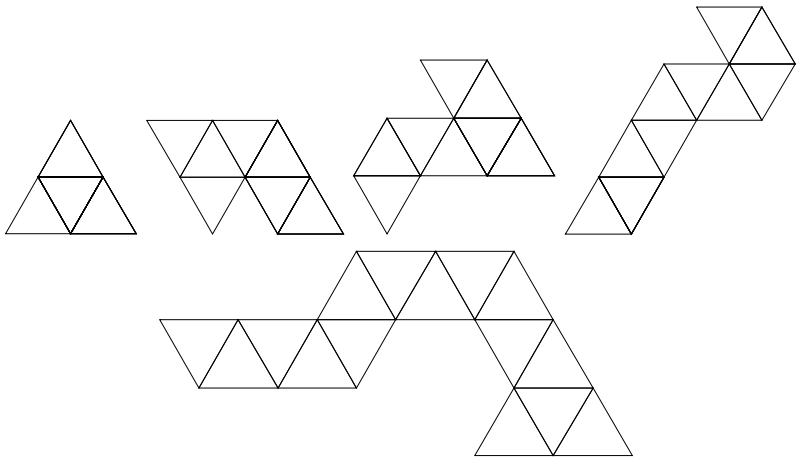


Fig. 2. Five solutions among 32768 to "15 triangles"

solution that is most consistent with the user's expectations, as we can see in [3] and [17]. The most common response to this problem is the use of heuristics to filter the results. When using a numerical method, the constrained figure is compared with each of the numerical solutions. This is generally characterized by slow runtimes, and there is often more than one solution left. Our formal approach allow us to take advantage of the construction program to compare the sketch with a solution.

Another concern of our team relates to software engineering, and deals with rigorous description of the models [7], and of their development [2]. That's why we used algebraic specifications formalism, and more particularly *OBJ3* language, to describe our geometric universe, and every notion we use. Notice that every notion used was specified using *OBJ3* and submitted to a series of tests. Another specification of analytic geometric universe thanks to algebraic techniques can be found in [14].

The rest of the paper is organized as follows. Section 2 outlines our formal approach of constraints solving, and the relating notions. Section 3 exposes our vision of likeness between figures, and a structuring of the solutions space. Section 4 presents a method to choose the intended figure amongst many solutions. Section 5 deals with restrictions that apply to it, and the way to solve particular cases. Section 6 shows some results provided by our technique,

and Section 7 concludes.

2 Geometric constraints system solving

Our original approach to formal geometric construction of rigid bodies in the Euclidean plane was made a reality with the prototype called YAMS. That is a formal 2D geometric solver associated with the 3D topology-based modeller *Topofil* [2]. A precise description of this association can be found in [18] and [9], so we'll only present here the solver part, which acts in two stages, a symbolic one and an interpretative one.

2.1 Symbolic resolution

In the first stage, given a dimensioned sketch, the solver associates the geometric objects with some identifiers, then turns the constraints into formal parameters to form the geometric constraint system as it is defined below.

Definition 1 (constraint system) *A* geometric constraint system *is a triple* $S = (U, X, C)$*, where* U *a set of parameters,* X *is a set of unknowns, and* C *a set of constraints of the form* $C = \{p_1(U, X), \ldots, p_r(U, X)\}$*, where each* $p_i(U, X)$ *is a predicative term, namely a constraint, whose variables are in* X *or in* U*.*

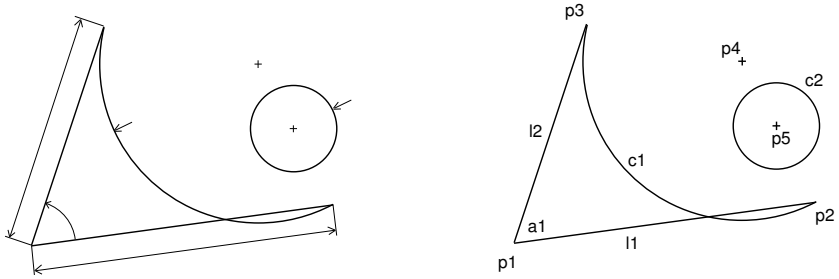


Fig. 3. A sketch with constraints (left hand side) and identifiers association (right hand side)

Example 2 *An example of dimensioned sketch is shown on the left hand side of Fig.3. The first object, made up of two line segments and an arc, is subjected to topological constraints (incidence and adjacency) deduced from the sketch, and metric constraints given by the user. As shown on the diagram, there are constraints on the two line segments lengths, on the oriented angle between these segments, and on the arc's radius. This arc is asked to have the same center as the circle that forms the second part of the figure. Finally, we want the radius of this circle to be half the radius of the arc. The sketch as it is*

drawn does not respect the metric constraints, but respects incidence and adjacency constraints. The right hand side of Fig.3 shows how YAMS associates identifiers to geometric objects. By convention, p_i, c_i, l_i, k_i, and a_i are the chosen names for points, circles, lines, lengths and angles, respectively. The corresponding constraint system is presented on Table 1.

Table 1
Constraints corresponding to Fig.4 example

egal_p(p5, p4)	angle(p1, p2, p1, p3, a1)	onc(p2, c1)
centre(c2, p5)	distpp(p1, p2, k2)	onl(p3,l2)
centre(c1, p4)	distpp(p1, p3, k1)	onl(p2, l1)
radius(c2, k4)	fixorgpl(p1, l1, p2)	onl(p1, l2)
radius(c1, k3)	onc(p3, c1)	onl(p1, l1)

When the user gives a dimensioned sketch, he actually provides YAMS some numerical values, for example *distpp(p1,p2,5)*, which are abstracted in flight to produce constraints, such as *distpp(p1,p2,k2)* and *k2 = initl(5)*. More precisely, the user provides an instanciated system $S_u = (\emptyset, X, C_u)$, where u is the tuple of the dimensions values, and YAMS transforms it into a system $S = (U, X, C)$, where each symbol of U is the abstraction of a dimension value, and C is the set of constraints given by the user where each dimension value is replaced by a symbol in U. It is possible to get back to S_u from S by instanciating symbols in U with values from u. An advantage of the formal approach is that changing the values of u does not affect the solving process. In the same way, in the particular case where all the constraints are metric constraints, that is involving dimensions, if u is the tuple of the values read on the sketch, then the sketch is a solution of S_u.

Then, according to the constraints, YAMS produces definitions of the form $y = g(u_1, \ldots, u_s, x_1, \ldots, x_k)$ that ensure correspondence between functional terms and identifiers. The definitions are brought together forming a general *construction plan*, which is the result of the formal phase. This plan indicates how and in what order the geometric objects must be built to produce the figure. More formally, a plan T is the result of a symbolic resolution of constraint system S if T is a triangular system, that is composed of definitions of the form $x_i = g(u_1, \ldots, u_s, x_1, \ldots, x_{i-1})$ where $1 \leq i \leq r$, r being the number of variables in T, and $S \equiv T$, that is S and T have the same solutions. We also say that T is a *solved* system. Using instanciation of parameter symbols in U with values of any tuple u, we obtain $S_u \equiv T_u$.

Example 3 *A construction plan corresponding to the constrained sketch of Example 2, that is where the parameters values were given by the user, is listed in Fig.4, on the left. In order to make this example clearer, we sum up some functional symbols and their profiles on Table 2.*

Table 2
Some functional symbols and their profiles

symbol	profile	comment
interlc	$line \times circle \rightarrow point$	intersection circle-line
mkcir	$point \times long \rightarrow circle$	circle with known center and radius
lpla	$point \times line \times angle \rightarrow line$	line through one point making a given angle with a known line
medradcir	$point \times point \times long \rightarrow circle$	circle passing through two points, with a known radius
intercc	$circle \times circle \rightarrow point$	intersection circle-circle

In addition, YAMS contains some original features that makes resolutions easier. The solver is able to break the initial geometric constraint system into smaller ones. This decomposition is a bottom-up process: the subsystems are discovered during the solving process. The philosophy is to solve subfigures independently and then to glue them together with a mechanism called *assembling*. Decomposition of the geometric constraint system is based upon the stability under displacements of dimensioned systems. YAMS uses a collaboration of several local methods, such as knowledge-based systems and Newton-Raphson method, coordinated by a multi-agent architecture with a blackboard. Geometric methods are fulfilled by expert agents which are production-based systems. A step by step process completes the blackboard with new pieces of knowledge until the problem is solved or no new deduction can be produced. An important hypothesis to ensure success is that the geometric constraint system to solve has to be well-constrained (see Section 1), that is it has a finite non-void set of solutions [9]. Actually, in this work, we don't deal with systems that are not well-constrained.

2.2 Interpretative stage

In the second stage, the required dimensions are used as parameters for the numerical interpretation of the construction plan. Since used functional terms may provide multiple results, each functional symbol is associated with a numerical *multifunction*. For example, the intersection between a line and a circle, symbolized by *interlc*, generally produces two points, and *medradcir* that builds a circle through two known points, with a known radius, generally produces two different circles (see Table 2). It is often useful to give a numbering to the various values produced.

Definition 4 (numbering) *Let g be a multifunction with n arguments and a maximum of k results. A* numbering *of g is a function G with $n+1$ arguments*

such that

$$g(x_1, \ldots, x_n) = \{G(x_1, \ldots, x_n, 1), \ldots, G(x_1, \ldots, x_n, k)\}$$

where $G(x_1, \ldots, x_n, i)$ *and* $G(x_1, \ldots, x_n, j)$ *are distinct functions of* $x_1, \ldots, x_n$ *if* $i \neq j$.

The existence of multifunctions in a construction plan introduces choices in the interpretation process. Once values are assigned to the parameters, we can consider the interpretation as the building of a tree labeled with numerical values. The interpretation of a definition of the form $y = g(u_1, \ldots, u_s, x_1, \ldots, x_r)$ produces a branching of degree k if multifunction g has a maximum of k results. At the end, the tree represents the solution space, and one solution corresponds to the labels of one branch. Note that during the evaluation, it may happen that a multifunction does not provide any result. In such a case, the interpretation stops in this branch. It may also happen that the number of the really produced results is less than the maximum. So we can distinguish the tree of potential solutions, called *tree of possibilities*, and the tree of effective solutions, called *tree of solutions*. Note that even if the tree of solutions is smaller than the tree of possibilities, it may increase fast and be very wide. Obviously, this tree is not really built, but explored by backtracking [18].

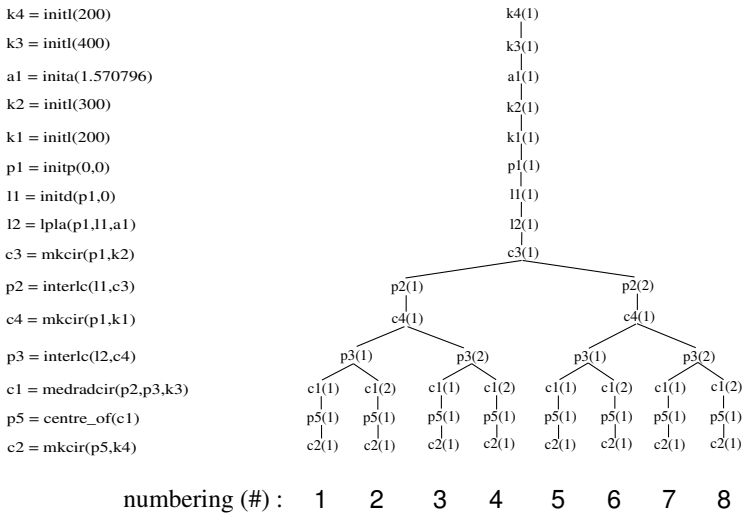


Fig. 4. Construction plan corresponding to Fig.3 and tree of solutions

Example 5 *A tree of solutions produced for our example after a parameters assignment is presented on Fig.4, at the right of the construction plan. The eight solutions (or branches) are shown on Fig.5, and numbered from* #1 *to* #8*. Each node corresponds to a result for an identifier, which is written with the result number in brackets.*

Thus, we have the following definition to precisely distinguish the solutions.

Definition 6 (occurrence of a solution) *Let T_u be an instance of a construction plan that defines the unknowns $x_1, x_2, \ldots, x_n$ thanks to the multifunctions $g_1, g_2, \ldots, g_n$ numbered by $G_1, G_2, \ldots, G_n$, with the maxima $k_1, k_2, \ldots, k_n$, respectively. A particular solution of T_u is $\Big(G_1(u, i_1), G_2(u, i_2), \ldots, G_n(u, i_n)\Big)$, where $1 \leq i_1 \leq k_1, \ldots, 1 \leq i_n \leq k_n$. We say that $(i_1, i_2, \ldots, i_n)$ is the* occurrence *of this solution.*

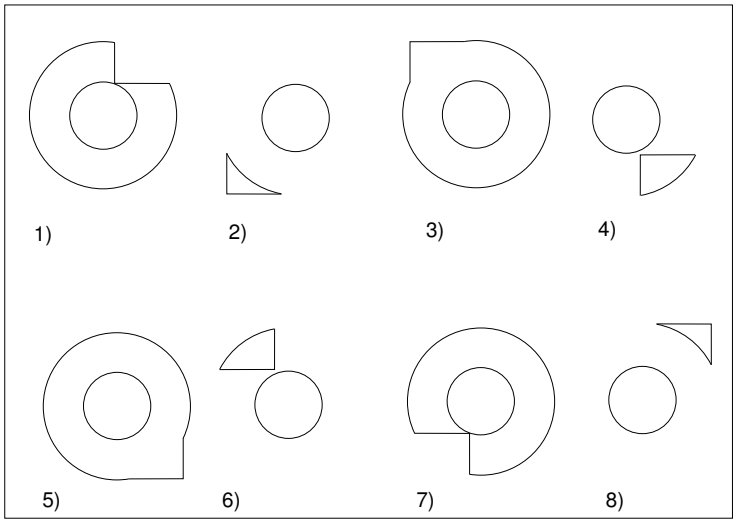


Fig. 5. The generated solutions

It is the usual notion of occurrence for tree nodes. For instance, one can read on Fig.4 that solution #6 has $(1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1)$ as an occurrence.

Actually, the computed construction plan enables to construct all the solutions as well as other figures which are "false solutions". The false solutions can quickly be eliminated thanks to a simple test, as they are not consistent with the constraints.

Example 7 *In our example, 4 solutions (numbered* 3*,* 4*,* 5 *and* 6*) can be eliminated because the sign of angle a_1 is the opposite of what is given in the constraints. Moreover, among the remaining solutions, we can eliminate* #7 *and* #8 *that are identical to* #1 *and* #2 *modulo a displacement, in the same way as in YAMS.*

But that may be insufficient. In the example presented on Fig.1, there are 32768 different solutions for a geometric object made of 15 equilateral triangles figure, but the solution space can't be reduced because all of the figures are consistent with the constraints. Other heuristics are necessary to drastically prune the tree of solutions, eliminating the figures that doesn't look like the sketch.

3 Tree of solutions pruning using sketch interpretation

Our purpose is to obtain a single solution figure that bears the best resemblance to the original drawing. Before explaining the method we use to achieve our aim, let us define what we mean by saying a figure looks like another.

3.1 Usual criteria of likeness

Similarity is defined by most of the dictionaries as conformity in nature or appearance between things. Two figures are often said to look like each other if they have some geometric properties in common, such as relative placing of points, lines and circles, angles acuteness, and convexity of some parts of the sketch. Conversely, two figures are not similar if one of the characteristics is satisfied by one and not by the other. This intuitive definition was proposed in order to eliminate solutions that seemed not "interesting" in the CAD framework. We only notice that most of these properties comparisons can be held in check by some simple examples: on Fig.6 all angles are acute and all points have the same relative placing so we can't decide which solution is required; on Fig.7 and 8 the sketch has a flaw (convexity or acuteness) with respect to the solution.

These intuitive criteria are not satisfactory. In the following section, a better criterion based on homotopy is proposed. Note that the verification of geometric properties on a sketch given by the user, called *semantic verification*, has already been used in the framework of geometric theorem proving [13].

3.2 Homotopy as a notion of likeness

In geometry, several notions of likeness between objects exist, depending on the considered abstraction degree. We try to define a criterion taking exactly

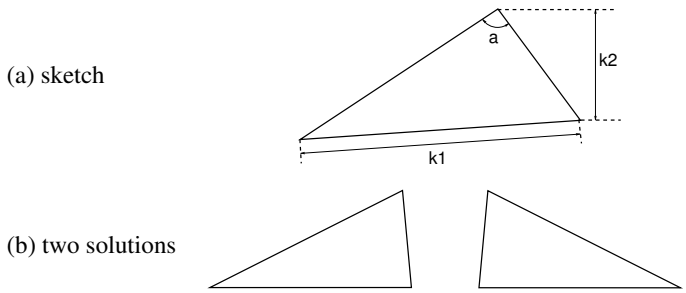
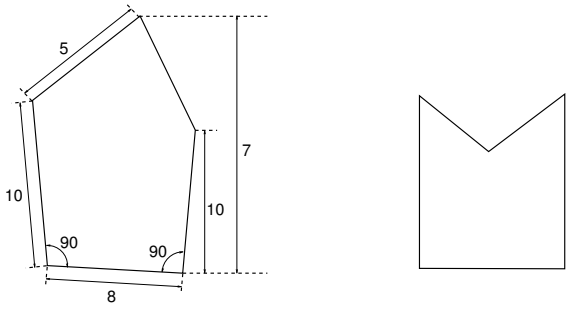
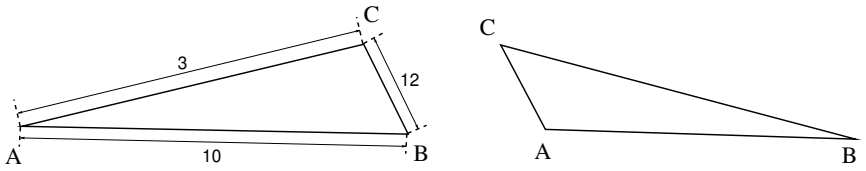


Fig. 6. Lack of discrimination criterion



(a) sketch (convex polygon) (b) a solution (concave polygon)

Fig. 7. Convexity flaw



(a) sketch (angle in C is obtuse) (b) a solution (angle in C is acute)

Fig. 8. Acuteness flaw

into account our framework, that is geometric figures which are solutions of constraint systems.

When considering only topological properties, the continuous deformation called homotopy is an usual likeness concept. Let us recall homotopy definition.

Definition 8 (homotopy) *Let $\mathcal{P}$ be a topological space, $f_0 : [0,1] \to \mathcal{P}$ and $f_1 : [0,1] \to \mathcal{P}$ be two parametric continuous curves. f_0 and f_1 are said* homotopic *if there is a function $\varphi : [0,1] \times [0,1] \to \mathcal{P}$ such that $\varphi(x,0) = f_0(x)$, $\varphi(x,1) = f_1(x)$, and φ is continuous.*

An interesting point of this definition is that it makes a bridge between a local notion of proximity in the Euclidean space and a global one. This aspect seems very interesting to us since in CAD the sketch can be very far from any solution (see Fig.6-8). Of course, this definition is much too general for us because it doesn't take into consideration elementary geometry properties of the objects, particularly their type. For example, a circle is homotopic to a triangle, and this is meaningless for CAD users. So, we have to refine this characterization in a geometric construction framework.

Let us first make clearer our notion of geometric type. In [9], we described a figure as a geometric tuple of objects, such as points, lines, circles, etc. The

type of such a figure is then a Cartesian product of simple types, to which we have to add incidence relationships. For example, the triangle ABC of Fig.8 is typed $point \times point \times point \times segment \times segment \times segment$ with the appropriate incidence constraints. With the help of a coordinates system, we can define a metric topology, from which a notion of proximity follows. Now we can give our definition of geometric homotopy.

Definition 9 (geometric homotopy) *Given two figures f_0 and f_1 of the same type $\tau_1 \times \tau_2 \times \ldots \times \tau_n$ with the same incidence relationships, we say that f_0 and f_1 are* geometrically homotopic *if there is a continuous transformation $\varphi : [0,1] \rightarrow \tau_1 \times \tau_2 \times \ldots \times \tau_n$ preserving incidence relationships, such that $\varphi(0) = f_0$ and $\varphi(1) = f_1$.*

Transformation φ of this definition can be seen as a tuple of continuous transformations $(\varphi_1, \ldots, \varphi_n)$ for the types $\tau_1, \ldots, \tau_n$.

Example 10 *On Fig.9, the figure composed of a triangle p1p2p3 and his circumcircle C is considered as having the type point × point × point × segment × segment × segment × circle. We deform the figure by translating p3 into p'3, that also deforms C into C'. The continuous transformations applied on the components are: identity on p1 and p2, translation $\overrightarrow{p3p'3}$ on p3, identity on [p1p3], similarities of centers p1 and p2 on [p1p3] and [p2p3] respectively, and a central homothety whose center is on the perpendicular bisector of [p1p2] on C.*

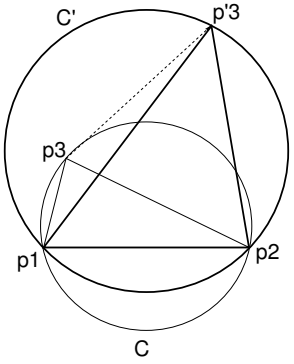


Fig. 9. Geometric homotopy

Note that the previous definition may characterize the interactive geometric deformations as in Cabri [23]. However, we don't simply deal with figures: we have to keep in mind the constraint system given by the user as much as the geometric figures. So we can't only consider a single solution, but its position in the entire solution space.

If we want to study continuous deformations of a dimensioned figure, not only have we to define the continuous deformation of a figure, but also of its constraint system. Both have to be subjected to a continuous deformation, from a particular solution of the system instanciated by some values u to another particular solution of the system instanciated by some values v. Actually, the notion of deformation of a system concerns the entire class of equivalent systems. Thus, we propose the following formalization.

Definition 11 (continuous deformation of a constrained system) *Let $S = (U, X, C)$ be a geometric constraint system instanciated into S_u and S_v with two tuples of values u and v for U. A* continuous deformation *from S_u to S_v is a continuous function $\psi : [0, 1] \to \mathbb{R}^s$ such that:*

(i) $\psi(0) = u$ *and* $\psi(1) = v$

(ii) for every system S' such as $S' \equiv S$, every well-constrained subsystem $\sigma' \subseteq S'$, and every $t \in [0, 1]$, $\sigma'_{\psi(t)}$ has as many solutions as σ'_u.

Recall (Section 2) that S and S' are said to be equivalent, denoted $S \equiv S'$, if they have exactly the same solutions. Point *(ii)* of Definition 11 is very strong, because it imposes that any subsystem σ' of the whole system S (or an equivalent system S') keeps the same number of solutions during the deformation. Indeed, once the parameters are fixed, say to u, the solution space can be viewed as the class of all systems equivalent to S_u. Thus, the above definition can be regarded as the continuous deformation of a solution space. The following lemma, directly coming from the definition, will be useful.

Lemma 12 *Let S and S' be two equivalent systems with the same parameters set, and u and v two instanciations of these parameters. If there is a continuous deformation from S_u to S_v, then there is a continuous deformation from S'_u to S'_v.*

This notion is linked with the notion of continuous geometric deformation of a figure the following way.

Definition 13 (S-homotopy) *Let S_u and S_v be two different instances of a constraint system S, f_u a particular solution of S_u, and f_v a particular solution of S_v. If the following conditions are satisfied*

(i) there is a continuous deformation ψ from S_u to S_v

(ii) there is a continuous function $\varphi : [0, 1] \to \tau_1 \times \tau_2 \times \ldots \times \tau_n$ such that $\varphi(0) = f_u$ and $\varphi(1) = f_v$

(iii) $\forall t \in [0, 1]$, $\varphi(t)$ *is a solution of* $S_{\psi(t)}$

then we say that there is a geometric homotopy from f_u to f_v with respect to the constraint system S, in short a S-homotopy.

In other words, the deformation of the constraint system must not reach any degenerate case because, if that occurs, we can jump to another solution instead of always following the same one, and that is not what we want.

Example 14 *On the top of Fig.10, the triangle ABC given as a sketch has three constraints: length BC equals k_1, distance between line $l_1 = (BC)$ and point A equals k_2, and oriented angle in A equals a. The two possible solutions, f_0 and f_1, are presented bottom left and right on Fig.10. When B and C are known, A can be built as the intersection between line l_2, that is at a distance k_2 from l_1, and the arc associated to a. It is possible to deform continuously the first solution by translating l_2, but if we pass through the degenerate case, as shown in the middle of Fig.10 (line tangency), then we can reach the second solution. Thus, f_0 and f_1 are not S-homotopic. We will explain later why the solution we would probably like to keep is f_1.*

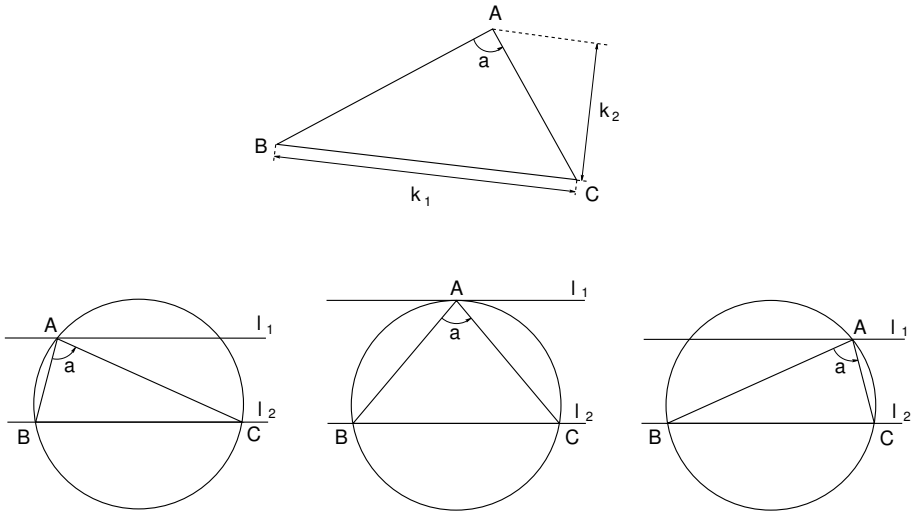


Fig. 10. Continuous deformation through a degenerate case

Since the symbolic resolution of a system S consists in building a triangular solved system T (see [9]) which is equivalent to S, the degenerate cases of S are the degenerate cases of T, and conversely. Given that, in our case, T is a set of definitions of the form $x_i = g(u_1, \ldots, u_s, x_1, \ldots, x_{i-1})$, where g is a multifunction, such a degenerate case occurs when g doesn't produce the maximum number of solutions, because of particular values of the parameters. For example, multifunction *interlc* (see Table 2) building the intersection between a line and a circle usually produces two solutions, but in the degenerate case where the circle is tangent to the line, there's only one solution (Fig.10). This leads us to give a numbering for multifunctions values which is compatible with the continuous deformation of the geometric constraint systems, and which we define as continuous numbering, this definition applying naturally to a construction plan.

Definition 15 (continuous numbering) *Let g be a multifunction and G a numbering such that*

$$g(x_1, \ldots, x_n) = \{G(x_1, \ldots, x_n, 1), \ldots, G(x_1, \ldots, x_n, k)\}.$$

Then G is a continuous numbering *if G is continuous on any domain containing no degenerate case.*

In all the following, it will be supposed that all the considered multifunctions are continuously numbered. Now that we defined clearly our vision of likeness and numbering, we are able to do the link between these notions, by expounding the following theorem.

Theorem 16 *Let S be a constraint system, and two instanciations u and v of the parameters such that there exists a continuous deformation from S_u to S_v and a triangular system solving S. When T denotes this triangular system, two figures f_u and f_v, respectively solutions of S_u and S_v, are S-homotopic if and only if they have the same occurrence $(i_1, \ldots, i_n)$ in T_u and T_v respectively.*

We do not prove this theorem here, let's simply say that the proof is mainly made of continuity arguments. The above theorem is true whatever the symbolic solution T of S, even if the occurrence is depending on T. More precisely, we have the following corollary.

Corollary 17 *Under the hypotheses of Theorem 16, two figures f_u and f_v, respectively solutions of S_u and S_v, are S-homotopic if and only if, for every triangular system T symbolically solving S, they have the same occurrence in T_u and T_v, respectively. In addition, f_u being a fixed solution of S_u, there is at most a unique f_v solution of S_v such that f_u and f_v are S-homotopic.*

3.4 Practical numbering

It is worth determining which geometric properties characterize the discrimination between the values of a multifunction. Following the example of *interlc*, let us consider the angle a between the given line and the line that passes through the center of the circle and one intersection in that order (see top left of Fig.11). Notice that lines and angles are oriented. In the degenerate case, a is a right angle. When *interlc* produces two solutions, they are discriminated by the acuteness or not of a, that allows us to produce a continuous numbering for *interlc*. For instance, solution number 1 will be obtained when a is acute, and solution number 2 when a is obtuse. Then, for every multifunction we currently use in our solver, we described such a geometric characteristic (see Fig.11 for illustrations):

- *interlc*: builds the intersection i between a line l_1 and a circle c. Let a be the angle between l_1 and the line l_2 that connects the center p of c and i. There are at most two solutions for this multifunction. For one solution a is acute, for the other one a is obtuse. There is a degenerate solution when c is tangent to l_1, and then a is a right angle.
- *intercc*: builds the intersection i between two circles c_1 and c_2. The two solutions are differentiated by the relative placing of three points (clockwise or counterclockwise): the centers p_1 and p_2 of the circles, and i. The degenerate solution occurs when c_1 and c_2 are tangent: p_1, p_2, and i are collinear.
- *mkcir4*: builds a circle c_2 tangent to a given circle c_1. One solution is located inside c_1, the other one is outside c_1. The degenerate case is reached when c_1 has a radius equal to zero.
- *medradcir*: builds a circle c through two points p_1 and p_2, knowing the radius k. Like *intercc*, the relative placing of three points (clockwise or counterclockwise) is different for the two solutions. In this case, the three points are p_1, p_2, and the center p of c. When $k = \frac{p_1p_2}{2}$, p becomes the middle of $[p_1p_2]$, the three points are collinear, and there is only one possible solution.
- *bisectdd*: builds the bisector l_3 of two lines l_1 and l_2. The angle a between l_1 and l_3 is either acute or obtuse. When l_1 and l_3 are parallel, we have a degenerate case.
- *linev*: builds a vertical line l at a given distance k from a point p. The latter is either at the left or at the right of l. If $k = 0$, then there is only one possibility for l, and it passes through p.
- *lineh*: builds a horizontal line l at a given distance k from a point p. The latter is either above or below l. If $k = 0$, then there is only one possibility for l, and it passes through p.
- *ldl*: builds a line l_2 parallel to another line l_1, at a given distance k from it. The angle a between l_1 and $\overrightarrow{p_1p_2}$, where $p_1 \in l_1$ and $p_2 \in l_2$, is negative or positive. If $k = 0$, then $l_1 = l_2$.

In each case, moving from a solution to the other consists in a continuous deformation passing throughout an intermediate figure where the value associated with the characteristic property reaches an extremum. All of the geometric properties defined above are preserved through a continuous deformation, so they allow us to define a continuous numbering.

4 Freezing of a branch

In order to find the solution that the designer expects, we put forward the hypothesis that the constraints he gave with the sketch reflect his expectations. More precisely, using Theorem 16 notation, we assume first that there is an instance u of the parameters such that the sketch f_u is a solution of S_u, and

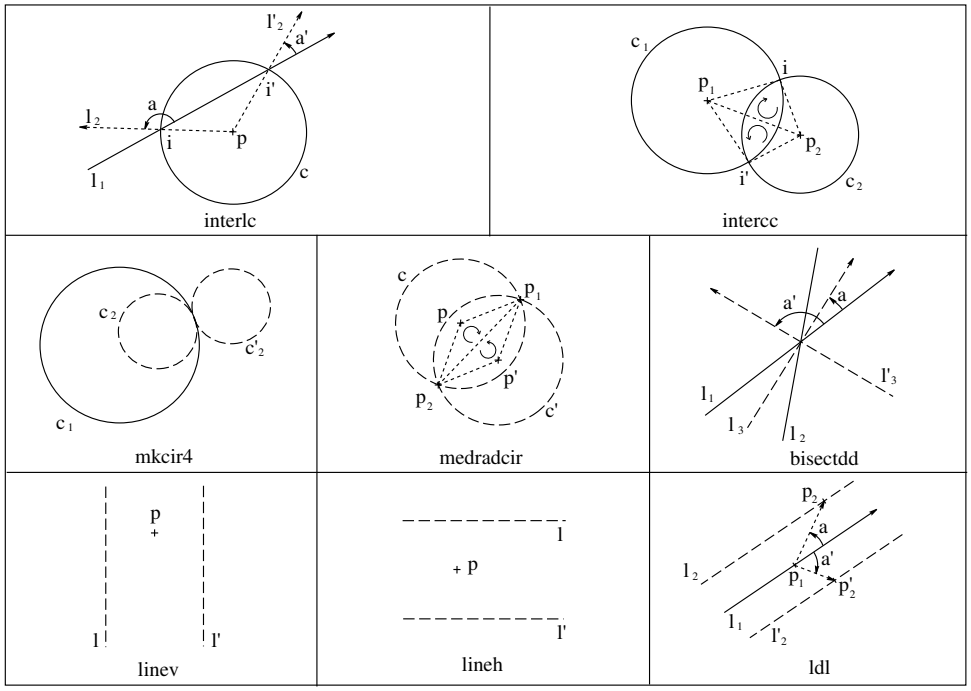


Fig. 11. Geometric characteristics of multifunctions

secondly that there is a continuous deformation from S_u to S_v where v are the dimensions values initially given in the constraints. In this section we explain how to fulfill the former condition in the case of metric constraints. The latter is a strong condition, but it seems to be usually achieved. For now, we will not consider peculiar cases, that will be discussed in the next section. That leads us to a simple way of finding the required solution, by applying Theorem 16 the following way.

We first point out the fact that, as we said in Section 2, the sketch can be seen as a particular solution of the constraint system S_u instanciated by the tuple u of the values read on the sketch. Then, we suppose that there is a continuous deformation between S_u and S_v, if S_v is the instanciation of the system by the tuple v of the given dimensions. Thus, if we can find a solution of S_v having the same occurrence than the sketch, then we can say that it is the intended solution. Even if it appears to be strange, we try to rediscover the sketch by the mean of the construction plan.

Practically talking, we take advantage of our formal solving approach by operating the following way. In a first stage, we apply the interpretation $\mathcal{I}_u$ to the triangular solved form T_u of the constraint system S_u with the sketch parameters. At each fork (multifunction) of the tree of solutions produced by $\mathcal{I}_u$, we can decide which branch to follow by comparing, as explained in Section 3.4, the geometric properties of the results of the multifunction with the data read on the sketch. The aim is that the computed version, that is the chosen

branch, has to be nearly identical to the effective sketch. That way, we can store the occurrence of this branch of the tree of solutions. This operation is called *freezing* of a branch.

In a second stage, for some fixed parameters v, the figure will be found by an interpretation $\mathcal{I}_v$ of the triangular solved form T_v of the constraint system S_v, simply following the branch that was frozen during the previous step and whose occurrence was stored. We can restart this interpretation with as many other parameters as we want without doing the freezing again.

An example of a result provided by this method can be seen on Fig.12. It shows the single solution found by using the freezing of a branch on the constrained sketch given on Fig.1.

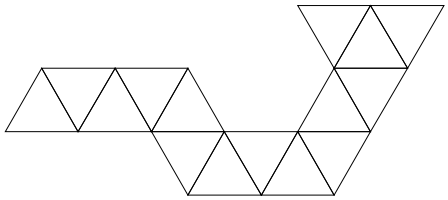


Fig. 12. The required solution of the sketch given on Fig.1

One of the advantages of this method is its speed, because no unnecessary comparisons are executed. Indeed, instead of geometrically comparing all the objects of the figure with each other, we only compare, at each junction of the tree, the objects that are brought into play in the concerned multifunction. Since the treatment is made as the interpretation goes along, this method reduces significantly the processing time in comparison with a systematic method.

Example 18 *In the example presented on Figs.1 and 12, it takes more than 1 minute to calculate all possible solutions, whereas our method gives an instantaneous good answer.*

Finally, as we said previously, the geometric criteria on multifunctions are dependent on lines orientation. For example, with *interlc*, if we change the line's orientation, the acuteness of the angle is inverted, and the solutions swap. So, in order to ensure numbering preservation, we must check that all lines in the sketch are given with the same orientation as the ones that will be computed. We have to compare the angle between each line and the Ox axis on the sketch with what it should be, as the sketch is considered as the result of an interpretation.

For example, let's consider a line l_1 described in the construction plan by a definition containing the function *lpla* that draws the line through one point p, making an angle α with another line l_2. We calculate, using the sketch's dimensions, what should be the orientation of l_1, knowing the previously cal-

culated orientation of l_2. Then, we compare this theoretical orientation with the effective one. If they are opposite, we change the sketch's data by reversing the line. On a formal point of view, the construction is unchanged. Only the numerical representation of the sketch is corrected.

5 Discussion

The method we exposed in previous section works fine when all constraints are metric. But another type of constraints is also used in YAMS: Boolean constraints. As examples, we can cite tangency, or equality of objects. In the case of Boolean constraints, some information is missing to find the intended solution. Actually, unlike metric constraints that don't affect topology, these constraints are not always respected on the sketch, as shown on Fig.13 and 14: on the sketch, the circle is actually not tangent to the line contrary to the constraints given by the user.

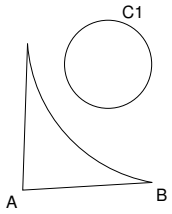


Fig. 13. Tangency problem: the sketch

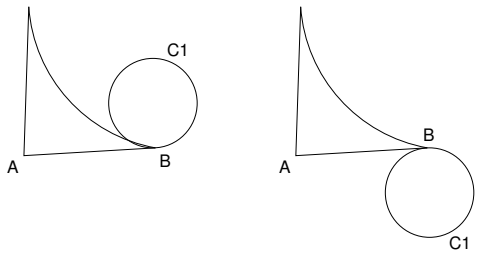


Fig. 14. Two possibilities for tangency

We notice that this kind of problem can come down to degenerate cases question. Indeed, tangency can be seen as a distance between a circle and a line which would be void, that is a limit case. In these situations, two general approaches can be considered, those correcting the sketch to fit the previous conditions, and those producing several branches giving a small subtree to explore.

The first approach has not yet been studied, but it may consist in interpreting the construction plan with the data read on the sketch by correcting within

itself the inconsistent objects as they go along. For example, on Fig.13, the sketch could be modified in such a way that the circle becomes tangent to the line. Among the two possible circles given by the construction plan, we choose the one that is closest to the circle on the initial sketch, in the sense of Euclidean distance over the coordinates.

The second approach consists in making a maximum freezing, that means keeping all the possible solutions if it is not possible to decide. The result is a *frozen subtree* that can be explored thanks to some tools provided by the software. These tools may be of different kinds. Let us enumerate some possibilities:

- automatic tools to prune and/or classify the branches of the remaining subtree. Classical geometric heuristics can be used such as comparisons of certain properties of the sketch and of the solutions. However, we take advantage of the construction plan to find out which objects are linked, and to compare their relative positions. This way, we think that these criteria are more pertinent. In our example above, the two possibilities for circle $c1$ can be sorted using the position (left or right) of the circle in the sketch.
- algorithmic tools to make faster the complete or the partial exploration of the solution space. Amongst these tools, which are well known in other domains, we have experimented a hash table, in the same way as functional languages like OBJ3, an intelligent backtracking, and some permutations of the construction plan to reduce the complexity [10]. We plan to use some heuristics coming from the SAT-problem [19,6].
- user friendly interface: we think that the user can have uncertain ideas and his/her wishes —expressed thanks to the sketch and the constraints— can be contradictory. So, we propose a user interface to explore the solution space using the construction plan. Let us detail these tools.

A first class of exploration tools is inspired by *debug tools* provided by most of the development systems in software engineering. This is possible because our approach is formal and we have a construction plan. So it is easy to do a step by step evaluation, allowing the user to choose, at each fork, a value among the available results. This simple mechanism can be enhanced with several kinds of breakpoint tools. Moreover, it is possible to offer the opportunity to freeze a part of a tree of solutions between two breakpoints, and then to skip this part that has become a big step.

We also intend to implement a second class of exploration tools, that is based on the idea of a *magnetic grid*. It allows a more intuitive approach of the selection problem. On the basis of a solution that doesn't fit the user's expectations, he can drag the misplaced element of the figure until one of the positions allowed by the tree of solutions.

6 Results

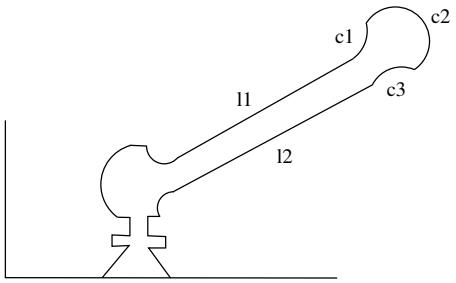


Fig. 15. Sketch of the lever

In order to illustrate our method, we expose here a quite representative example. The sketch on Fig.15, showing a lever, comes with 106 constraints including 2 tangency constraints, $tgcl(c1, l1)$ and $tgcl(c3, l2)$ (see Table 3). Note that to lighten the figure, we avoided to represent the constraints by arrows, as we did in the previous examples.

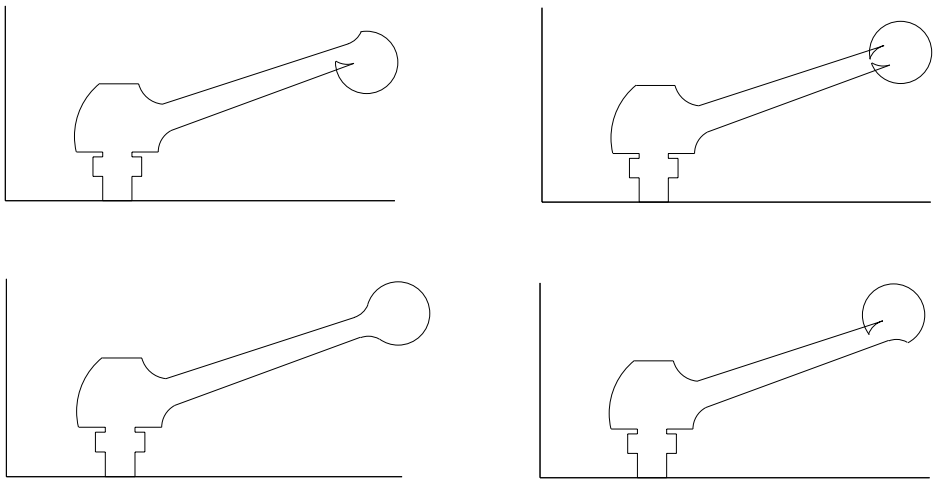


Fig. 16. Solutions for the lever

Using these constraints, that we cannot explain in detail in such a paper, the solver produces a construction plan, containing 252 definitions. As among these definitions, 29 have an arity of 2, the tree of possibilities of this constraint system has $2^{29} = 536870912$ branches. Actually, some branches lead to an interpretation failure, and others can be eliminated with a simple constraints verification, so the tree of solutions provided by the interpretation stage only has 160 branches.

Our method allows to prune significantly the tree of solutions, providing a four-branched subtree. The remaining solutions can be seen on Fig.16. Among the

Table 3
Constraints of the lever

angarc(p28, p3, p25, a18)	angarc(p26, p24, p2, a17)	centre(c6, p31)
centre(c4, p29)	centre(c3, p28)	centre(c2, p27)
radius(c5, k31)	radius(c6, k30)	radius(c4, k29)
centre(c5, p30)	centre(c1, p26)	radius(c2, k28)
radius(c3, k27)	radius(c1, k26)	angle(p13, p16, p13, p11, a16)
angle(p17, p16, p17, p18, a14)	angle(p15, p16, p15, p12, a13)	angle(p18, p17, p18, p19, a12)
angle(p10, p12, p10, p9, a10)	angle(p19, p18, p19, p20, a9)	angle(p20, p21, p20, p19, a8)
angle(p21, p20, p21, p22, a6)	angle(p8, p9, p8, p7, a5)	angle(p22, p21, p22, p23, a4)
angle(p16, p17, p16, p15, a15)	angle(p12, p15, p12, p10, a11)	angle(p9, p10, p9, p8, a7)
angle(p17, p14, p4, p3, a2)	angle(p17, p14, p1, p2, a1)	angle(p5, p6, p22, p23,a3)
distpp(p11, p13, k25)	distpp(p13, p14, k24)	distpp(p15, p16, k20)
distpp(p13, p16, k23)	distpp(p16, p17, k22)	distpp(p18, p17, k21)
distpp(p12, p15, k19)	distpp(p19, p18, k18)	distpp(p19, p20, k17)
distpp(p12, p10, k16)	distpp(p9, p8, k12)	distpp(p5, p8, k8)
distpp(p21, p20, k15)	distpp(p9, p10, k14)	distpp(p21, p22, k13)
distpp(p23, p22, k11)	distpp(p7, p8, k10)	distpp(p5, p6, k9)
distpp(p7, p5, k7)	distpp(p1, p22, k6)	distpp(p1, p23, k5)
distpp(p14, p4, k3)	distpp(p4, p3, k2)	distpp(p1, p2, k1)
distpp(p23, p4, k4)	fixorgpl(p13, l10, p16)	onc(p7, c5)
onc(p1, c6)	onc(p6, c6)	onc(p5, c5)
onc(p4, c4)	onc(p23, c4)	onc(p3, c3)
onc(p25, c2)	onc(p24, c2)	onc(p24, c1)
onc(p25, c3)	onc(p2, c1)	onl(p21, l16)
onl(p23, l17)	onl(p22, l17)	onl(p22, l16)
onl(p21, l15)	onl(p20, l15)	onl(p20, l14)
onl(p19, l13)	onl(p18, l13)	onl(p18, l12)
onl(p17, l10)	onl(p16, l10)	onl(p16, l11)
onl(p15, l9)	onl(p14, l10)	onl(p13, l10)
onl(p12, l9)	onl(p12, l7)	onl(p11, l8)
onl(p10, l6)	onl(p9, l6)	onl(p9, l5)
onl(p8, l4)	onl(p7, l4)	onl(p6, l3)
onl(p4, l2)	onl(p3, l2)	onl(p2, l1)
onl(p19, l14)	onl(p17, l12)	onl(p15, l11)
onl(p13, l8)	onl(p10, l7)	onl(p8, l5)
onl(p5, l3)	onl(p1, l1)	tgcl(c1, l1)
tgcl(c3, l2)		

four figures, the part that remains unchanged corresponds to the metric constraints, and the uncertain part corresponds to the two tangency constraints. It is not possible to have only one solution since the tangency constraints are not respected on the sketch. To prune the subtree, we have implemented other heuristics such as relative placing of circles and lines. With these heuristics, we found the intended solution, that is the bottom-left part of Fig.16.

7 Conclusion

In this paper, we exposed our formal approach of geometric constructions, that yields a construction plan from a dimensioned sketch. Then, we defined a notion of likeness coming from the topological homotopy notion called *S-homotopy*. This allows us to define in some way what is the structuring of the

solution space of a constraint system. We also proposed a method to select *one* solution amongst many, by *freezing* a branch of the tree of solutions with the help of the sketch.

This method works fine while all constraints are metric, as we shown on some simple examples. However, as the limits of our method were clearly identified, we studied some tools to explore the tree of solutions, and to help the user to find the intended solution.

In previous papers [9], we exposed that a symbolic solving has many advantages. Here we showed that it is also useful for solutions selection or exploration. Various debugging tools will soon be implemented, and we plan to develop an intuitive graphic interface to deal with them.

Further work is needed to analyze more in detail the structuring of solutions space. For example, in the case of articulated systems animation, one of the problems is the crossing of dead points. This problem is linked with some degenerate cases, and with the transition from one branch to another in the tree of solutions.

References

[1] B. Aldefeld, H. Malberg, H. Richter and K. Voss. Rule-based variational geometry in computer-Aided Design. *Artificial Intelligence in Design*, D.T. Pham editor, Springer-Verlag, p. 27-46, 1991.

[2] Y. Bertrand, J.F. Dufourd. Algebraic specification of a 3D-modeller based on hypermaps. *Computer Vision - GMIP*, 56(1):29-60, 1994.

[3] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer-Aided Design*, 27(6):487-501, 1995.

[4] B. Brüderlin. Using Prolog for constructing geometric objects defined by constraints. *Proceedings of EUROCAL'85*, LNCS 204, Springer-Verlag, p.448-459, 1985.

[5] B. Brüderlin. Automatizing geometric proofs and constructions. *Proceedings of Computational Geometry'88*, LNCS 333, Springer-Verlag, Berlin, p.232-252, 1988.

[6] L. Brisoux-Devendeville, C. Essert-Villard, and P. Schreck. Exploration of a solution space structured by finite constraints. *14th European Conference on Artificial Intelligence, Workshop on Modelling and Solving Problems with Constraints*, F:1-18, 2000.

[7] J.-F. Dufourd. Algebras and formal specifications in geometric modelling. *The Visual Computer*, 13:131-154, 1997.

[8] J.-F. Dufourd, P. Mathis, and P. Schreck. Formal resolution of geometric constraint systems by assembling. *Proceedings of the ACM-Siggraph Solid Modelling Conference, Atlanta*, p.271-284, 1997, ACM Press.

[9] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99:73-119, 1998.

[10] C. Essert, P. Schreck, and J.-F. Dufourd. Interprétation d'un programme avec multifonctions géométriques. *Proceedings of 6èmes Journées de l'AFIG, Dunkerque, France*, p.223-232, 1998.

[11] L.W. Ericson and C.-K. Yap. The design of Linetool, a geometric editor. *Proceedings of Computational Geometry'88*, LNCS 333, Springer-Verlag, Berlin, p.83-92, 1988.

[12] J.X. Ge, S.C. Chou and X.S. Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31:867-879, 1999.

[13] H. Gelernter, J.R. Hansen, D.W. Loveland. Empirical explorations of the geometry theorem proving machine. *Computer and Thought*, MacGraw Hill, p.134-152, 1963.

[14] J.A. Goguen. Modular algebraic specification of some basic geometrical constructions. *Artificial Intelligence*, 37:123-153, 1988.

[15] G.A. Kramer. A geometric constraint engine. *Artificial Inelligence*, 58:327-360, 1992.

[16] V.C. Lin, D.C. Gossard, and R.A. Light. Variational geometry in CAD. *ACM Computer Graphics (SIGGRAPH'81)*, 15(3):171-175, 1981.

[17] H. Lamure and D. Michelucci. Solving constraints by homotopy. *Proceedings of the ACM-Siggraph Solid Modelling Conference*, ACM Press, p.134-145, 1995.

[18] P. Mathis. Un système de résolution de contraintes par assemblage en modélisation géométrique, Ph.D. Thesis, Université de Strasbourg, 1997.

[19] B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathemathics and Artificial Intelligence*, 22:319-331, 1998.

[20] J. Owen. Algebraic solution for geometry from dimensional constraints. *Proceedings of the 1st ACM Symposium of Solid Modelling and CAD/CAM Applications*, p.397-407, 1991, ACM Press.

[21] P. Schreck. Implantation d'un système à base de connaissances pour les constructions géométriques. *Revue d'Intelligence Artificielle*, 8(3):223-247, 1994.

[22] J.M. Scandura, J.H. Durnin, W.H. Wulfeck II. Higher order rule characterization of heuristics for compass and straight edge constructions in geometry. *Artificial Intelligence*, 5:149-183, 1974.

[23] H. Schumann and D. Green. *Discovering Geometry with a Computer using Cabri Geometre*, Chartwell-Bratt, 1994.

[24] I.E. Sutherland. Sketchpad: A man-machine graphical communication system. *Proceedings of the IFIP Spring Joint Computer Conference*, p.329-36, 1963.

[25] G. Sunde, A CAD System with Declarative Specification of Shape. *in Intelligent CAD Systems I - Theoretical and Methodological Aspects*, P.J.W. ten Hagen, T. Tomiyama eds, EurographicSeminars, Springer-Verlag, p. 90-104, 1987.